Amendment to the Title:

Please replace title with the following amended title:

~~Using Incremental Generation to Develop Applications~~ Using Incremental Generation to Develop Software Applications

The above title appears in the Application in the Cover Sheet and on the first page of the Application on the first line.

The full text of the title shall now read:

Using Incremental Generation to Develop Software Applications

Amendments to the Specification:

Please replace paragraph starting on page 8, line 21 and ending on page 9, line 3 with the following amended paragraph:

As mentioned above, the development process in system 100 is based on a metamodel. FIG. 2 illustrates a metamodel 200 (e.g., a semantic information model), which is represented using a unified modeling language ~~(UML)~~("UML™") class diagram, in conjunction with a portion 205 of a development process for developing a user interface application 260. The specifications for ~~UML~~UML™, which are set forth by the Object Management Group (OMG), can be found on OMG's web site ~~at http://www.omg.org/uml~~.

Please replace paragraph starting on page 9, line 4 and ending on page 9, line 15 with the following amended paragraph:

As illustrated in FIG. 2, the classes of metamodel 200 can include application components 220, views 225 with UI elements 230, controllers 235 with context nodes 240 and context attributes 245, and models 250 with model classes 255. Components 220 represent reusable entities that can be used to group various application elements (e.g., models, views, controllers, and contexts). Context node 240, which is associated with controller 235, provides a storage area for the data used by the controller 235, as well as methods for storing and accessing the data based on a declared data structure. The metamodel 200 also can include relationships between the metamodel classes, as shown in FIG. 2. The ~~UML~~UML™ class diagram of the metamodel 200 can be viewed, in essence, as a formal description of the programming model for a user interface

application. An advantage of using ~~UML~~UML™ to represent metamodel 200 is that the representation of the metamodel is platform-independent.

Please replace paragraph starting on page 10, line 14 and ending on page 10, line 23 with the following amended paragraph:

FIG. 3 illustrates a ~~UML~~UML™ class representation of another metamodel 300. The classes of metamodel 300 include a component 305, which is associated with a model 310. In ~~UML~~UML™, an association represents a semantic relationship between two classes that involves a connection between the instances of those classes. Model 310 includes an aggregation of zero or more model classes 315. In ~~UML~~UML™, an aggregation is a form of an association that represents a whole-part relationship between an aggregate (the whole) and the constituent part(s). The open diamond of an aggregation association is located at the aggregate class. Model class 315 includes an aggregation of zero or more model class attributes 320. Model 310 also includes an aggregation of zero or more model relations 325. Model relation 325 includes an aggregation of two instances of the class model relation role 330.

Please replace paragraph starting on page 11, line 1 and ending on page 11, line 15 with the following amended paragraph:

In addition to the association with model 310, component 305 includes an aggregation of zero or more views 335 and zero or more controllers 340. View 335 includes an aggregation of a User Interface (UI) element 345. As illustrated by relationship 350, UI element 345 can include an aggregation of other UI elements 345. Controller 340 includes an aggregation of a context node 355. Context node 355 includes an aggregation of zero or more context attributes 360. As illustrated by relationship 365, context node 355 also can include an aggregation of other context nodes 355. Context

node 355 is associated with model class 315 and model relation role 330. Context attribute 360 is associated with model class attribute 320. Context element 370 is a general class from which either of the specialized classes context node 355 or context attribute 360 are derived. In ~~UML~~UML™, a generalization shows an inheritance relationship between objects. The hollow arrowhead points to the general class (e.g., superclass). UI element context binding 375 is associated with context element 370. In an implementation example, this represents that UI element context binding 375 can be associated with context node 355 or context attribute 360. UI element 345 also includes an aggregation of zero or more UI element context bindings 375.

Please replace paragraph starting on page 12, line 5 and ending on page 12, line 11 with the following amended paragraph:

Special features using the customizable extensions of ~~UML~~UML™

The portions of the metamodel shown in FIG. 3 illustrate both standard ~~UML~~UML™ constructs and customizable extensions, such as stereotypes. Using these customizable extensions, the metamodel can be extended beyond the standard ~~UML~~UML™ definitions to add additional features and functionality to the derived metadata API 130, thereby including customized features within API 130. These features enable API 130 to be more than just a development layer for creating and persisting development objects.

Please replace paragraph starting on page 12, line 12 and ending on page 12, line 20 with the following amended paragraph:

One set of features uses the customizable extensions stereotype and tagged value of ~~UML~~UML™ to customize the metamodel classes. For example, one feature is to use a stereotype <<mdo>> for a class. Using this stereotype indicates that the class is a main development object (MDO) of a metamodel. A main development object indicator specifies that instances of the marked class (i.e., the class marked with the <<mdo>> stereotype) are persisted into one XML file together with any non-MDO children objects. Use of this feature enables the metamodel designer to determine where the file borders are located with respect to the persisted development objects, which can be used to determine the granularity of certain features performed at the main development object level.

Please replace paragraph starting on page 13, line 23 and ending on page 14, line 10 with the following amended paragraph:

Model 403 includes model classes customer 411a, order 411b, order item 411c, and product 411d (generally model classes 411). Model classes 411 are illustrated in FIG. 4 using a ~~UML~~UML™ representation. Model class 411c includes an attribute 413. Attribute 413 is labeled "Qty." and is an integer type. Controller 409 includes context nodes context 415a, order node 415b, item node 415c, and product node 415d (generally context nodes 415). Each context node 415 includes one or more context attributes. Context node 415a includes attributes 417a and 417b. Context node 415b includes attributes 419a (labeled "ID"), 419b, and 419c. Context node 415c includes attributes 421a (labeled "No."), 421b (labeled "Qty."), and 421c. Context node 415d includes attribute 423 (labeled "Name"). The attributes are generally referred to as attributes 417, 419, 421, and 423. View 406 includes a display layout 425 that includes UI elements 428a, 428b, and 428c (generally UI elements 428).

Please replace paragraph starting on page 17, line 12 and ending on page 18, line 8 with the following amended paragraph:

To help provide an example of how generator 140 determines which development objects to re-generate, FIG. 5 illustrates a generic portion 500 of an application with related main development objects 505, 510, 515, and 520. Each main development object 505, 510, 515, and 520 includes non-MDO development objects. For example, main development object 505 includes development objects DO_1.1, DO_1.2, DO_1.3, and DO_1.4. As described above, in some examples a main development object represent a file border for instances of so designated ~~UML~~UML™ classes, where the main development object instance includes within the file instances of non-MDO children and ancestor development objects. Following these examples, each MDO 505, 510, 515, and 520 corresponds to a different file. As described in more detail below, generator 140 uses these file borders to determine which MDOs to re-generate. MDOs 505, 510, 515, and 520 are related to each other by aggregation relationships (e.g., 525 and 530) and association relationships (e.g., 535, 540, 545, and 550, shown as a dashed line). It is worth noting that some of the relationships between MDOs are a result of relationships directly between two MDOs, such as aggregation relationship 525. Other relationships, however, are a result of relationships between non-MDO development objects, such as association relationship 535, which is between development objects DO_1.1 and DO_2.2 and/or a result of relationships between a non-MDO development object and a MDO, such as relationship 530, which is between development object DO_3.2 and MDO 520. As described in more detail below, relationships are followed along any of these paths to determine relationships between MDOs.